Truth and Breadth, Clarity and Depth in Algebra

Neville Holmes, *University of Tasmania*

The formulator does to the calculator what the calculator did to the abacus.

In an essay titled "Truth and Clarity in Arithmetic" (The Profession, Feb. 2003, pp. 126-128), I outlined a simple calculator design that avoided the several unfortunate faults in the commodity calculator. Although I got some laudatory e-mails, I was bemused by one from a calculator designer who told me I had no idea how a calculator should be designed, but who failed to point out any specific fault in my design.

Since then, digital technology has brought in devices like the iPod and BlackBerry, about the size of the commodity calculator, which makes it interesting to consider what might be done to the calculator to exploit their technology. I will call this extended design a *formulator* and hope thus to avoid blanket condemnation.

THE EXACT CALCULATOR

The design of the exact calculator was motivated by the unmet need, especially in early education, for truth and clarity, and is the basis for the formulator's design.

Truth was mainly achieved by providing only exact arithmetic. Combined integer and fractional arithmetic was thus the basis, and this required a notation that allowed both decimal and other fractions to be represented exactly. While exactitude ruled out functions such as the square root, and values such as multiples of π, it also made desirable very simple functions such as quotient and remainder.

Clarity was mainly achieved by requiring a minimum of four lines of display so that at least the two or three numbers involved in the immediately prior calculation, and a number being keyed in for the next, would be clearly visible. Also, as befits a calculator, the tapping of a function key caused the calculation with that function to be carried out, and the result and its sources displayed together with the function symbol. In this context, a function is literally an operation.

The representation of numbers was enriched by the use of two symbols: $^{\nabla}$ for the negative sign and fraction point and $_{\Delta}$ for the decimal point—provided within the usual three-by-four digital key matrix, with four basic function keys alongside. The number of distinct functions provided was greatly expanded by being able to use the two special symbols as prefixes to the basic function symbols, and to use all functions as either monads or dyads.

TRUTH

The next step up from exact arithmetic is inexact arithmetic, and the challenge is to stay truthful.

Truth in this case resides in proclaiming inexactitude for numbers both on the way in and on the way out. Italic representation, with a shift key for it, is one way of meeting this need. And of course exactness is maintained if possible, at least internally, and the greatest practical accuracy assured otherwise.

Handling inexactness makes many extra functions useful, such as for exponentiation and trigonometry, thus handling imaginary and complex numbers. An interesting notation for this would use the $ symbol for the imaginary point, so that 2$3 would represent the number traditionally and confusedly represented as an arithmetic expression: *2+3i*. Nondecimal values such as dates, times, and angles—represented maybe as *2009y10m19 7w6d5h12*

Continued on page 106

Published by the IEEE Computer Society

Continued from page 108

76d54m32—have interest but need special treatment in the arithmetic.

Many useful functions can be represented by symbolically modifying basic functions. In the exact calculator, the two special symbols were used for this, but a larger keyboard allows a larger set of distinctive symbols to be used, and modifiers can be placed like accents above the function symbol. Thus monadic $\tilde{\times}$ squares its argument, while dyadic $\tilde{\div}$ reverses its arguments, dividing its first argument *into* its second.

Much of the handling of lists can be done by providing an edit capability, for example to extend or combine lists or to replace, add, or remove items. But when a result depends on the values within the list, a formal functional approach works best, with structural functions alongside but distinct from arithmetic ones. Such functions strictly preserve the values of list items so that different kinds of numbers can be mixed within a list.

The simplest notation for combining functions is juxtaposition.

Another way to augment a function is to use an integer as superscript to have the function repeatedly applied, so that $\times^2$ multiplies its first argument by the square of its second. A zero superscript leaves the first or only argument unaltered while a negative superscript inverts the function, so that monadic $\tilde{\times}^{\nabla 1}$ ÷ takes the square root of its argument.

BREADTH

To add breadth, the formulator works on lists of numbers. If a simple arithmetic function has two list arguments, then they must be of the same length. A dyadic function with one argument a list and the other a single item applies the single item to each item of the list.

Lists of numbers are awkward to display, particularly on a handheld device, so a suite of graphical display options is used to help the user. Keying lists of numbers in can also be awkward, but abbreviation conventions help. For example, subscripted replication as in H_2O and CO_2 allows 99_{10} as a list of ten 99s, and 6[7]89 provides integers between 6 and 89 spaced by 7. The subscript is useful on display as monadic ≅ immediately shows the number of items in its argument, at least if there is neither an infinity $1^{\nabla}0$ nor an indeterminacy $0^{\nabla}0$.

Arithmetic functions can change the list's structure, however. The acute accent signals the arithmetic reduction of a list, so that monadic $\acute{+}$ would total a list while dyadic $\acute{+}$ would add a list up in groups of a size given by the single item argument. Monadically, $\acute{-}$ gives the alternating sum and $\acute{\div}$ the alternating product.

Arithmetic functions can produce more than one valid result from a single item—such as analytical functions that extract the factors of an exact value or the roots of a complex value or a polynomial—by allowing items of a list to be sets. This adds a good deal of simple richness to the arithmetic, especially for students. Notationally, a set is enclosed in parentheses, and converting a list to a set removes duplicates and puts the items in sequence.

CLARITY

Thus far, the formulator works like an operational calculator in that one or two arguments are selected and a single function, simple or modified, then operates on the arguments to produce an immediate result. This means that complex calculations are procedurally complex and their nature hidden behind that complexity.

Clarity is achieved by providing for functions to be combined notationally, as in traditional algebra. Operationally, this means maintaining two stacks: one for potential arguments and another for potential functions. Editing can be done in either stack, and then any calculation is a kind of anticlimax to developing an algebraic function, testing it, and eventually applying it. Calculation is done by selecting one or two arguments from their stack, then selecting their function from the formula stack.

The simplest notation for combining functions is juxtaposition, in the same way that digits are juxtaposed to form numbers. All functions in such a compound are used monadically except the lowest-order one, which is used whichever way the compound function is used. Thus $\tilde{\times}\div$ is monadically the square of the reciprocal of its argument, and dyadically the square of the first argument divided by the second. If part or all of a compound function is to be modified, it is enclosed in parentheses with the modifier placed over one parenthesis.

A basic compound function applies each component successively to the result coming from its right. Only the rightmost function of a compound sees the compound's argument(s). However, the decimal-point symbol $_\Delta$ is also used in compound functions as a dyadic point. Functions to the right of the $_\Delta$ are monadically applied to each argument, with their results joined by the first function to the left of the point. Thus dyadic $+_\Delta\div$ is the sum of the reciprocals of its arguments.

The next level of notation is the list of functions, called a *train*. Items in the train can be simple or compound functions, and parentheses can be used to enclose a train to make it an item within a compound or train.

A train of two items is called a *hook*; its first item is a dyad and the other a monad. The second argument of the dyad is the result of the monad that is applied to the second

or only argument. The first argument of the dyad is the first argument of a dyadic hook, or the only argument of a monadic hook. So monadic + ÷ adds the argument to its reciprocal.

A train of three items is called a *fork*; its center item is dyadic with its arguments the results of its neighboring items. The neighboring items each take the argument or arguments of the fork. Thus monadic +́ ÷ +́≡ calculates the mean of its argument, though the formulator would probably have a primitive symbol for the +́≡ compound.

In a longer train with an odd number of items, the first two items form a fork with the rest of the train. In a train with an even number of items, the first item forms a hook with the rest of the train.

DEPTH

Clarity in the formulator is supported by the ability to construct formulas by putting functions together in various ways to define a sequence of evaluation. For depth, such construction is supported by templates used to select members of a family of functions in a process of "contemplation."

A template uses placeholder symbols to define where in the template one or two figments will be used. Figments are to a template much like what arguments are to a function. Templates are kept in the function stack but cannot be used directly for calculation.

A new function is produced by selecting as figments one or two functions from the function stack or keyboard, then selecting the template to be used. The new function joins the function stack.

THE FORMULATOR

The design I have outlined is constrained in several important ways. First, the numerical data for calculation are either items or lists of items, where an item can be a set but not a list. This is not as limiting as it might seem. A dyadic function with arguments of different length—to calculate a polynomial for a list of arguments, for example, can be taken to need its second argument dribbled into the function item by item to be worked on by the function with its entire first argument to reduce it to a single item of the result. The ~ modifier can be used to have the items of the first argument dribbled in instead.

Second, the notation is entirely symbolic, free of alphabetic characters. This is inspired by the thoughts of the late great Kenneth Iverson, as expressed in his Turing Award essay, "Notation as a Tool of Thought" (elliscave.com/APL_J/tool.pdf), which describes the principles behind his APL (A Programming Language).

Iverson later surrendered to the tragic typographical tyranny of the computer industry and its profession and led a redesign of APL called J (jsoftware.com) based on the ASCII character set and the QWERTY keyboard. Incidentally, the idea of trains is used in J and occurred to Iverson as he flew back to Canada from an APL conference in Sydney, Australia.

APL, J, and several related systems are splendid for programming. However, the formulator is not designed for programming. The commodity calculator is a device for ordinary people and students to use for ad hoc calculation. Similarly, the formulator, as a commodity device, is for ordinary people to use for ad hoc algebra, and for students to use to learn algebra.

The decline of literacy and numeracy is well attested. Digital technology can be used in early education to counteract this (The Profession, Mar. 2008, pp. 102-104), and this is starting to happen.

But the decline in numeracy and thence the study of mathematics in later education continues (see tinyurl.com/ye5q5yc, for example). This can't be counteracted by the use of mathematical packages like APL and Mathematica in schools because their proper use must be based on an understanding of the mathematics involved. An expert user of such packages is not necessarily an expert mathematician.

What is needed is a mathematical tool like the formulator. Development of the necessary standard for such a tool, and training teachers to use it, is an important challenge to the computing and mathematics professions. C

Neville Holmes *is an honorary research associate at the University of Tasmania's School of Computing and Information Systems. Contact him at neville.holmes@utas.edu.au.*